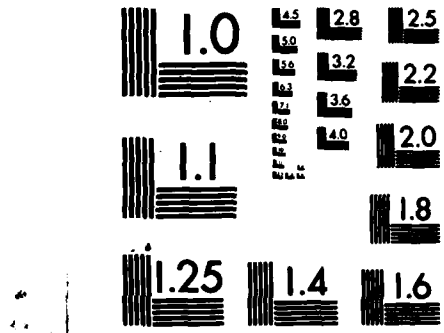
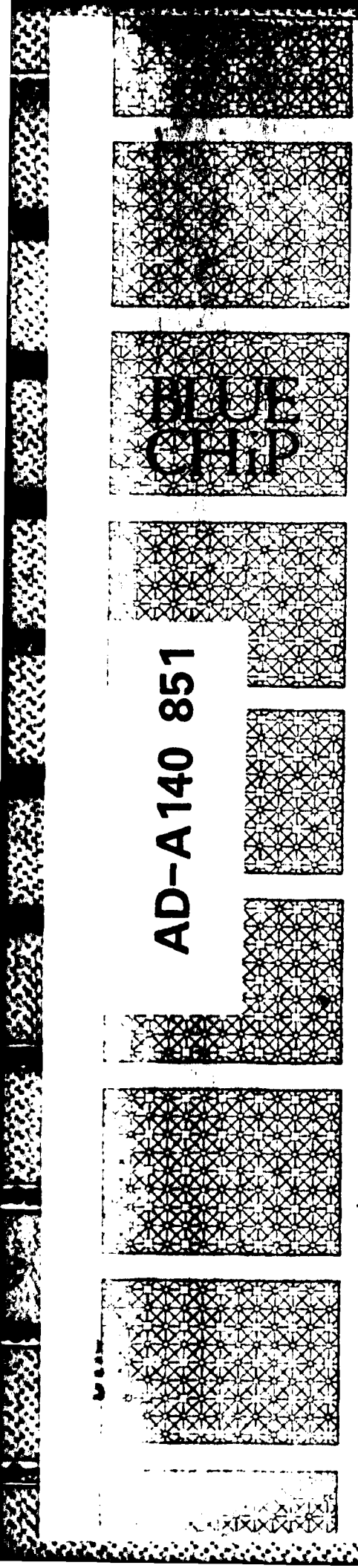
AD-A140 851 INTRODUCTION TO THE POKER PARALLEL PROGRAMMING ENVIRONMENT(U) PURDUE UNIV LAFAYETTE IN DEPT OF COMPUTER SCIENCES L SNYDER AUG 83 CSD-TR-434 1/1

UNCLASSIFIED N00014-80-K-0816 F/G 9/2 NL





(12)

Introduction to the Poker Parallel Programming Environment

by

Lawrence Snyder

The BLUE CHiP Project
Purdue University
Department of Computer Sciences
Math Sciences Building
West Lafayette, Indiana 47907

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CSD-TR-434	2. GOVT ACCESSION NO. A140857	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) INTRODUCTION TO THE POKER PARALLEL PROGRAMMING ENVIRONMENT		5. TYPE OF REPORT & PERIOD COVERED Technical, interim 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Lawrence Snyder		8. CONTRACT OR GRANT NUMBER(s) N00014-80-K-0816 N00014-81-K-0360
9. PERFORMING ORGANIZATION NAME AND ADDRESS Purdue University Department of Computer Sciences West Lafayette, Indiana 47907		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Task SRO-100
11. CONTROLLING OFFICE NAME AND ADDRESS Office on Naval Research Information Systems Program Arlington, Virginia 22217		12. REPORT DATE August 1983 13. NUMBER OF PAGES 4
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

DTIC ELECTE S MAY 7 1984 D B

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

parallel programming, parallel software, parallel operating system, Poker, XX, CHiP Computer, programming environment, lattice

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Poker Parallel Programming Environment is a graphics-based, interactive system for programming the Configurable, Highly Parallel (CHiP) Computer. Designed to support nearly all aspects of parallel programming in one integrated system, Poker has been implemented as a (35,000 line) C program on the VAX 11/780 under UNIX. It provides a number of novel features including graphics programming of parallel processor communication.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Introduction to the Poker Parallel Programming Environment

by

Lawrence Snyder
Department of Computer Sciences
Purdue University
West Lafayette, Indiana 47907

CSD-TR-434

Introduction to the Poker Parallel Programming Environment

Lawrence Snyder

Department of Computer Sciences
Purdue University
West Lafayette, Indiana 47907

ABSTRACT

The Poker Parallel Programming Environment is a graphics-based, interactive system for programming the Configurable, Highly Parallel (CHiP) Computer. Designed to support nearly all aspects of parallel programming in one integrated system, Poker has been implemented as a (~35,000 line) C program on the VAX 11/780 under UNIX. It provides a number of novel features including graphics programming of parallel processor communication.

Although much sequential programming can be accomplished with only the support of a programming language compiler, loader and run-time system, parallel programming is too complex to be done with such rudimentary facilities alone. The Poker System is an interactive programming environment to support the Configurable, Highly Parallel (CHiP) Computer [1]. The Poker System is not itself a parallel program, but rather it is a "front end", implemented on the VAX/780 under UNIX. It is a front end to a preprototype version of the CHiP hardware, called the Pringle, which is a 64 processor parallel computer emulating the CHiP [2]. In addition, Poker is a front end for a complete software emulator for the Pringle.

The Poker System enables the programmer to define a large family of CHiP architectures with 4 to 4096 processors. Programs can be written and emulated for any family member. Facilities are provided to define processor interconnection structures graphically, to program the processing elements, to compile, coordinate [3], and load, to perform single and multistep execution, and to peek and poke (whence the name) at the memory of the architectures.

CHiP Programming is Something Else

The programming environment provided by Poker is somewhat unconventional due partly to novel properties of the CHiP Computer and partly to novel properties of the system itself. To increase the accessibility of subsequent sections, we discuss here the *activity* of CHiP programming and the *role* Poker plays.

Programming, of course, is the conversion of an (abstract) algorithm that is "machine independent" into a form suitable for execution on a particular computer. Thus, to begin programming a CHiP machine, we need to have a parallel algorithm in mind. The algorithm is presumed to have the form of a graph whose vertices are processes and whose edges specify the communication paths among the processes.

This work is part of the Blue CHiP Project. It is supported in part by the Office of Naval Research Contracts N00014-80-K-0816 and N00014-81-K-0360. The latter is Task NR0-160.

For example, Figure 1 gives an algorithm that uses a binary tree as the communication graph. The algorithm finds the maximum of a set of numbers (stored one per process in a local variable called "val") and then multiplies each number by the maximum. The maximum is found by "floating" the largest value in each subtree to the root of that subtree. Then the global maximum is broadcast back through the tree where each process multiplies it times its local "val." Notice that although there are fifteen processes in the tree, there are only three *types* of processes used.

The conversion of this algorithm to run on a CHiP computer, *i.e.*, the programming, is straight forward.* It involves

(a) embedding the communication graph into the switch lattice,

(b) programming the process types in a sequential programming language,

(c) assigning one of the process types to each processor,

(d) naming the data path ports, and

(e) compiling, assembling, coordinating, and loading the program.

We consider each of these activities in turn.

Embedding the communication graph into the switch lattice requires that we program the switches of the lattice so that the processors have a topology that matches (or is a super set of) the topology of the communication graph. This embedding operation is done graphically (rather than symbolically) in the Poker System using the Switch Settings mode. Figure 2 illustrates a particular embedding of the fifteen node binary tree into the lattice. Processor (1,2) is the root of the processor tree, processor (1,1) is a leaf, and processor (1,3) is unused.

Next we program the three process types in a sequential language, XX. Each process is viewed as a procedure with (optional) parameters and local variables. In addition to the usual declarations we must specify the *port names*, symbolic names used by a process to refer to other processes with which it communicates. Figure 3 shows the XX code for the three process types. In the programs the symbol '<-' is used for input/output; assigning to a port name, e.g., PARENT <- val, causes output, and assigning from a port name, e.g., max <- PARENT, causes input.

The construction of the processor tree in the switch lattice to match the communications graph gives an implicit association between the processes of the algorithm and the processors. We make this relationship

*Assuming familiarity with the CHiP Computer [1].

0190-3918/83/0000/0289$01.00 © 1983 IEEE

explicit by assigning process names to the appropriate processors using the Code Names mode of the Poker System. Figure 4 gives the result.

Next, the port names mentioned in each process must be associated with a specific data path. Each processor has eight ports corresponding to the compass points. Only those connected by an active data path to another PE need be named. This activity is performed using the Port Names mode of Poker. Figure 5 shows the result of naming the ports.

The algorithm is now programmed. Next, each process type mentioned in the Code Names specification is compiled into assembly code. The assembly code is then "coordinated," i.e., modified so that the CHiP Computer can run it synchronously. The coordinated programs are assembled to produce processor object code. The interconnection structure is "compiled" to produce switch object code. The object codes are loaded into the machine and executed.

Description of the Poker Environment

In the last section we used the Poker Programming Environment to embed graphs, to define processes, to assign processes to processors, and to declare port names. The discussion implied the existence of certain facilities in the Poker System. Now we give a more complete description of those facilities.

The Poker System is an interactive programming environment that uses two displays: The primary display is a high resolution (1024 × 768 pixel) bit-mapped display, and the secondary display is a conventional character display. The two displays are used to increase the amount of information available to the programmer. Most activity takes place on the primary display; XX programming is usually done on the secondary display.

The primary display has the form illustrated in Figures 2-5. The bottom square region, called the *field*, is where most of the programming activity takes place. The field always displays some schematic representation of the two dimensional array of processors being programmed. The exact form of the representation changes depending on whether the programmer is performing a graph embedding, a process assignment, a port declaration, etc. Since the field is not always large enough to show the whole schematic representation, a map of that portion being displayed is given in the upper left-hand corner. Status information, diagnostics and miscellaneous data are given in the upper right region of the display, called the *chalkboard*. The bottom line of the chalkboard is the *command line*, used for specifying the few textual commands* required by Poker, such as reading library files.

The logical structure of the Poker Environment is shown in Figure 6. It provides an integrated set of facilities to

- define architecture characteristics (CHiP Parameter),
- embed communication graphs (Switch Settings),
- program process code segments (XX language),
- assign processes to processors (Code Naming),
- declare port names (Port Naming),
- compile, coordinate, assemble and load (Command Request),
- execute, trace, peek and poke (Trace Values).

We now describe each of these facilities in detail.

Architectural definition. Because Poker is intended to be a laboratory tool for studying CHiP programming, it has been designed to support a number of CHiP family architectures. Programs can be written for logical CHiP machines with from 4 to 4096 processors. All of these logical machines can be emulated using a software emulator, and one family member, the 64 processor version, will be able to be run on a hardware emulator, the Pringle [2], when it is completed. Consequently, the programmer begins using Poker by specifying the characteristics of the underlying logical architecture. These include the number of processing elements and the amount of routing capability needed for the lattice (corridor width [1]). The default parameters are those that match the machine defined in the previous session, or, if there was none, then the parameters of the Pringle hardware.

Graph embedding. The field of the primary display shows the lattice of the current architecture, as illustrated in Figure 1. The activity is largely that previously described; the programmer connects the processors (represented as boxes) with line segments to define edges. Graphics primitives based on cursor keys permit edges to be drawn and erased. Facilities are available for following graph edges, managing the display (e.g., centering), saving embeddings, reading in library embeddings, etc.

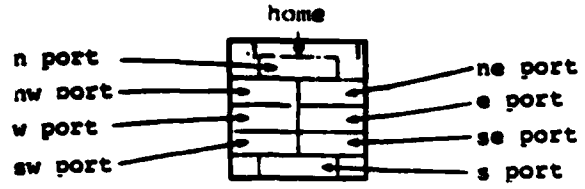
Programming the process code segment. The XX (Dos Equis) sequential programming language is a simple scalar language for defining processes. The language has four data types (Boolean, character, integer and real), the common control structures (while, for, if-then-else, etc.), vectors and the usual supply of scalar arithmetic and logical operators. In addition to data type declarations, one can also declare scalar variables to be port names, procedure parameters, or variables to be traced. Input/output is performed by assigning from or to a port name. The semantics are "data-driven:" writes occur immediately and reads wait on the arrival of data, if necessary. XX process codes are generally developed on the secondary display using a standard editor.

Process assignment. The processors are assigned processes using a field display on the primary terminal like those in Figure 4. The programmer enters the name of the process procedure on the first line of the processor box. If the procedure has formal parameters, then values for the actual parameters can be entered on the following (four) lines. Facilities are provided for buffering the contents of a box and then automatically depositing the contents of the buffer into processors in whole regions of the processor array. In this way the programmer is saved from manually entering repeated information when the algorithm exhibits uniformity.

Port declarations. The field of the primary display has the form illustrated in Figure 5. Each processor has up to eight incident edges as a result of the graph embedding, and it has been assigned a process which refers to up to eight port names. These are matched

*Poker is completely interactive; most actions are given as a single key stroke and have immediate effect.

using the port declaration. The processor box is divided into eight windows:



The programmer enters the names used by the assigned process code into the window for that edge. The names are clipped to the first five characters. Facilities are provided for displaying unclipped names in the chalkboard, and like the process assignment, it is possible to buffer port assignments and deposit them automatically in whole regions of the processor array.

Program translation. The preceding facilities provide a means of specifying the elements of a Poker program. They are then converted into executable form. The XX compiler converts each process to assembly code. The coordinator [3] then attempts to convert the process assigned to each processor into a form that permits the entire program to run with synchronous (i.e., not data-driven) execution. [This step can be by-passed and the processes can be run in data-driven form.] If coordination is successful, the processors may all have different assembly codes associated with them. In any event the assembly converts the assembler code to object form. The connector "compiles" the graphical representation of the communication graph into an object form. The object code and the object graph as well as the actual parameter values are loaded into the emulator (or the Pringle).

Execution. The resulting program is executed and the traced variables are displayed; the field is similar to that used for process assignment. The execution can proceed for a given number of steps, or until a displayed value changes. When the execution is suspended, any of the displayed values can be changed. When execution resumes these new values are poked back into the processor memory.

Further detail about the Poker Environment can be found in the references [4,5].

References

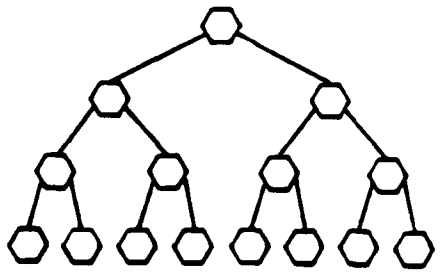
[1] Lawrence Snyder
Introduction to the Configurable, Highly Parallel Computer
Computer, 15(1): 47-56, January 1982

[2] J. Timothy Field, Alejandro A. Kapauan, and Lawrence Snyder
Pringle: A Parallel Processor to Emulate CHiP Computers
Technical Report CSD-TR-433, Purdue University, 1983

[3] Janice E. Cuny and Lawrence Snyder
Compilation of Data-driven Programs for Synchronous Execution
Proceedings of the 10th Symposium on the Principles of Programming Languages, ACM, pp. 197-202, 1983

[4] Lawrence Snyder
Parallel Programming and the Poker Environment
Technical Report CSD-TR-443, Purdue University, 1983

[5] Lawrence Snyder, Steven S. Albert, Carl W. Ampert, Brian G. Benning, Alan J. Chester, John P. Guaragno, Christopher A. Kent, John Thomas Love, Eugene J. Shekita, Carleton A. Smith
The Poker Programming Environment and its Implementation
Technical Report CSD-TR-410, Purdue University, 1982



leaf process:
write val to parent;
read max from parent;
val ← val · max;

ancestor process:
read x from left child;
read y from right child;
write max (x, y, val) to parent;
read max from parent;
write max to left child;
write max to right child;
val ← val · max;

root process:
read x from left child;
read y from right child;
max ← max (x, y, val);
write max to left child;
write max to right child;
val ← val · max;

Figure 1. An algorithm; each leaf is an instance of the leaf process, the root is an instance of the root process and all other nodes are instances of the ancestor process.

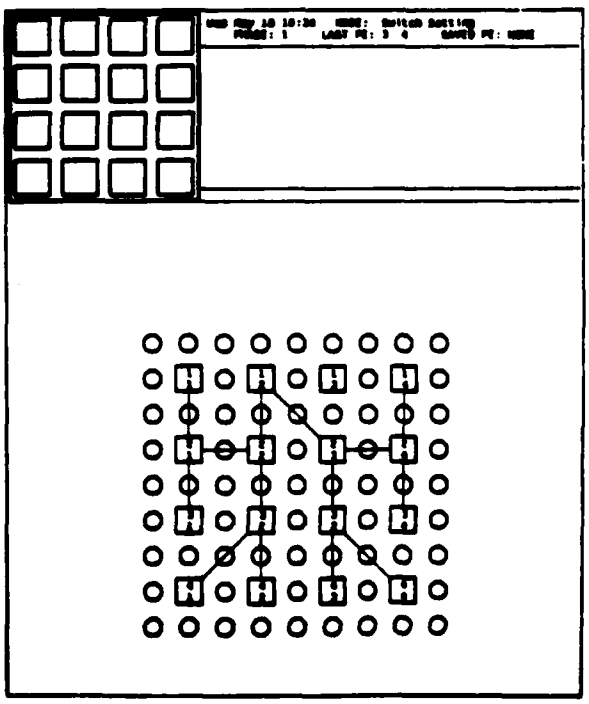


Figure 2. An embedding of the 15 node binary tree.

```
code leaf (val);
ports PARENT;
begin
int max. PARENT;
PARENT <- val;
max <- PARENT;
val:=val * max;
end.

code root (val);
ports LCHILD. RCHILD;
begin
int x.y. max. val.
  LCHILD. RCHILD;
x <- LCHILD;
y <- RCHILD;
if x> y then max:=x
  else max:=y;
if val > max then max:=val;
LCHILD <- max;
RCHILD <- max;
val:=val * max;
end.

code ancestor (val);
ports PARENT.LCHILD.RCHILD;
begin
int x.y. max. val.
  PARENT. LCHILD. RCHILD;
x <- LCHILD;
y <- RCHILD;
if x> y then max:=x
  else max:=y;
if val > max then max:=val;
PARENT <- max;
max <- PARENT;
LCHILD <- max;
RCHILD <- max;
val:=val * max;
end.
```

Figure 3. Code for the three process types.

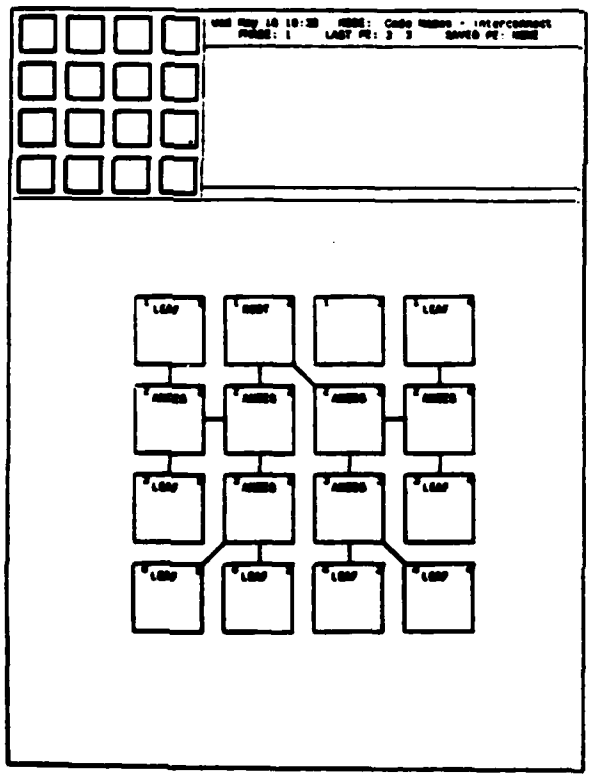


Figure 4. Assignment of process names to processors; note that the name "ancestor" has been clipped to five characters.

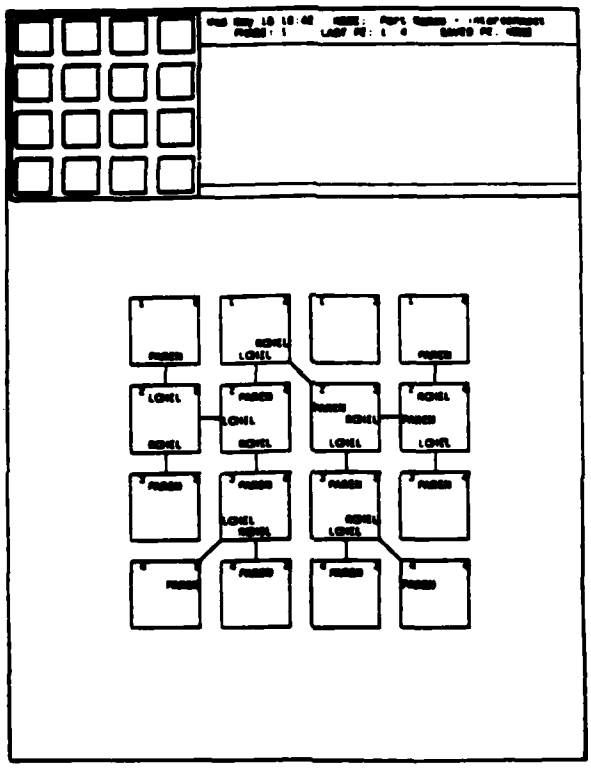


Figure 5. The specification of the port names; note that the names have been clipped to the first five characters.

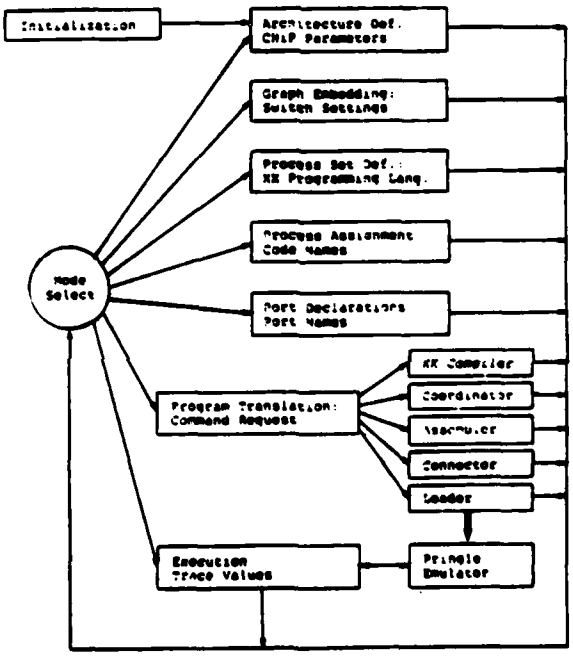
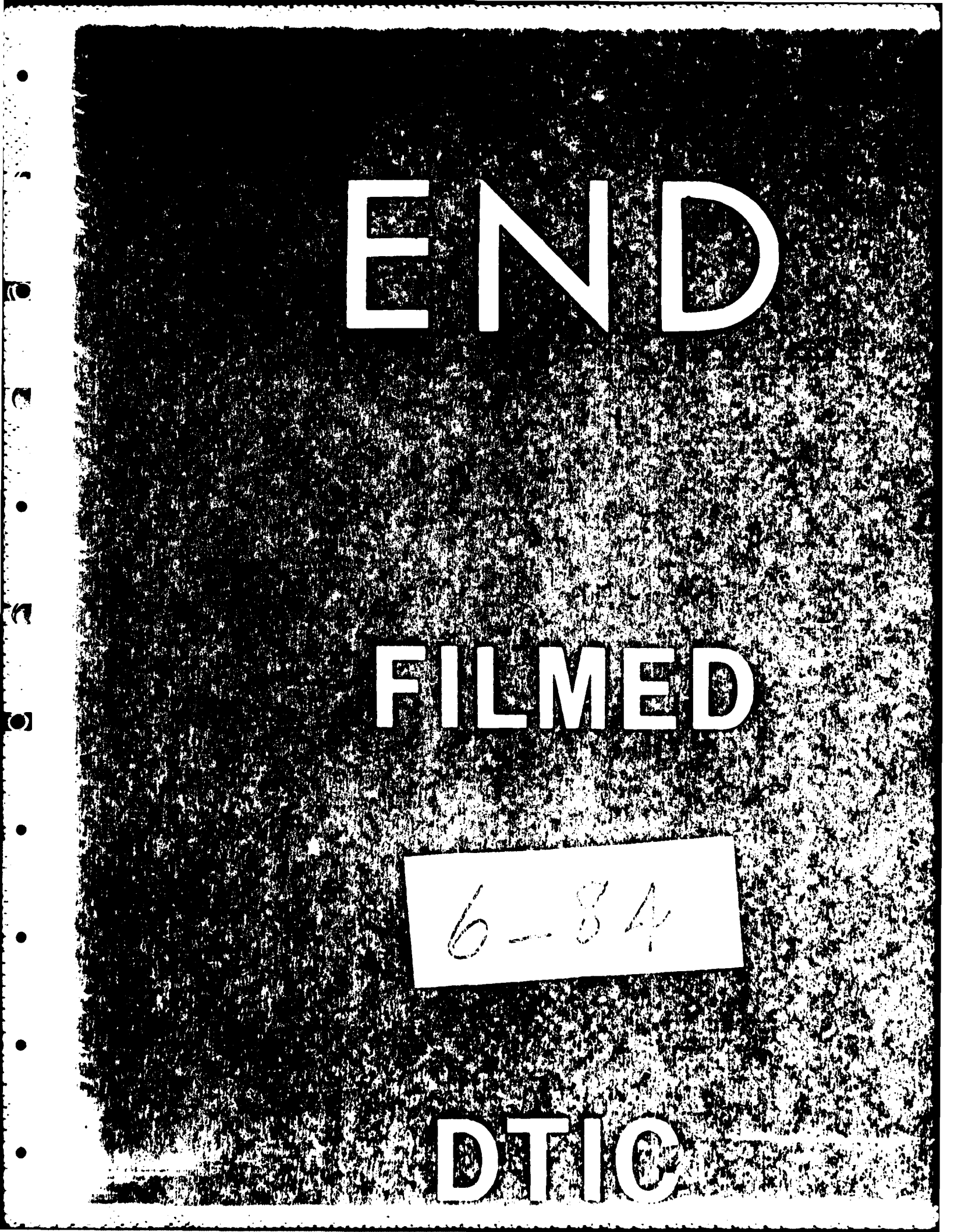


Figure 6. The logical structure of the Poker Environment.


END
FILMED
6-84
DTIC